ReportGeneration

<?xml version="1.0" encoding="utf-8"?>
<html>

Statistics eporting

Once we are collecting logs in whatever form is best, we also need to give consideration to the best way of producing reports on those logs. I feel that where possible we should be generating flat XHTML reports periodically. This is for a number of reasons: first it will keep load on the user facing part of the application down, also it will allow the implementors (i.e. the people customising the report generation) the opportunity to define consistent and rich reports without spending time performing either searches or custom queries on the data.
ased on the reporting that I did for DStat, the following seem like likely useful report sections:
a) **An executive summary**. Management like these, as they provide quick and easy advocacy information for repositories and high level overviews of the activity. This would consist of a configurable list of actions to provide aggregation information on. Probably we would want to be able to break this down into per instance, per community and per collection reports.
b) **Archive content analysis**. This would not draw from the logs, but from the state of the archive at analysis time. At the moment I envisage this working similarly to DStat, where a list of content types could be provided to report on. This would also be useful at a community and collection level as well as a whole instance level.
c) **Item views listing**. This would show the usage of items on a per instance basis. Again, reports for collections and communities could produce this information in their own reports.
d) **Full action report**. asically a list of all the actions that were performed, and the number of times. Also available at community and collection level reports
e) **Most useful reports**. This would be an implementer configured list of the actions that they may be most interested in. This is effectively an extended Executive Summary. Also available at community and collection levels
f) **Averaging Information**. This would provide vague averages of usage of things. e.g. average number of times that items in the repository have been accessed. Also, this could include averaging information on administrative statistics: e.g. average amount of time an item spends in workflow. This may be useful in a community and collection report also.
g) **Full individual item reports** - This would be a full audit trail for activities regarding the item. It would include administrative and activity statistics.

Detailed section reports

eporting on the values of parameters on general reports would be both unnecessary, ungainly, and really really hard to layout nicely. Therefore it would make sense to have each action linked to a more specific report on that action's parameters alone. The general reports would then look like:
{{

```
Unknown macro: {Action | Count | Full Info----------+-------+----------Licence ejected | 27 | <a>detailed view</a>Search | 1003 | <a>detailed view</a>}}
```

}
And an individual page concerning all the parameters passed by, for example, a "search" action could be provided on one page. It is at this point that having the control list of parameters as discussed in other pages might come in handy, providing a basis for comparison and aggregation and so forth.

Analysis processing

While having a database driven log system would allow for all sorts of dynamic querying, I am still in favour of having periodically generated files. These could be much richer than administrator generated queries, since we can spend time both designing them and executing them. To this end I have a vague outline of a framework that might be appropriate for managing logs.
attachment:stats.png
You can see that all mediation with the database is done via the LogAnalyser class (except the ContentAnalyser, which is a slightly different sort of beast). I am suggesting that the LogAnalyser provide the interface to the actual log data for all report generating classes. Each actual analyser is then a tool which knows how to produce reports for its particular purposes. For example the ActionAggregator knows simply how to count the different actions performed and produce a series of different action summaries in XML. The ItemAnalyser does all the work required to build item audit trails. The ActionAnalyser does something slightly more complicated than the ActionAggregator by creating individual reports for each carried out action. Later, at the UI level, the outputs of the ActionAggregator and the ActionAnalyser would need to be linked together as mentioned above. All reports should be generated in XML, which needs ideally to be standardised, which will make it easy to transform them into whatever display mediums are required (XHTML for the UI, and perhaps other XML formats for stored audit trails)
The ContentAnalyser is slightly different in that it does not require access to the system logs. Instead its job is to inspect the archive contents and make calculations as to, for example, item types held.
A further thought suggests that we may want to allow individual modules to provide their own analysis tools to allow them to report on specific functionality. So to execute a log analysis I imagine we use the eport class with some parameters, which in turn calls each of the individual analysers within the confines of some framework which would generate reports for the relevant periods, produce the XML output, and use the Transformer classes to implement an XSLT transform into the desired XHTML output. Exactly how this works is still a little hazy, but it would seem sensible if each analyser could have its own subdirectory in, say, /dspace/reports for XML reports, and there would be a general directory for rendered report pages. There is probably a necessity for some kind of reports servlet which helps maintain the linking between different reports (like I say, hazy!).

An example XML document for storing report results

{{

```
Unknown macro: {<report start="<start date>" end="<end date>" name="<report name>"><title>The title of the report section}
```

}
This would make it relatively straightforward to build tables of results which could be as big or small as necessary. ut is it really enough information ...

Some notes

- Floor values are common to cut off reporting at a level where the information is either not useful, or would result in an excess of unnecessary data. One observation I made since putting DStat together is that the floor values which are good for the general reports are not good for the monthly reports. Therefore if we use flooring (which we should), we will want to implement floor values on a per-report type basis (e.g. general, monthly, weekly, daily, per instance, community or collection) for every report type that will have a floor reporting value.
- ScottYeadon showed us some nice graphs that Cocoon was putting together from SVGs that were being generated from their statistics. Can we easily include this sort of functionality without necessarily using Cocoon (e.g. using apache batik inline?)
- What other forms of statistics might we need? Please add your thoughts. Here are some additional suggestions for customised analysis:
- Most common search words
- Most active user
- Community/Collection audit trails
- There will need to be some kind of access control to the statistics, especially if there will be sensitive data included, such as personal email addresses. DStat dealt with this by anonymising email addresses if configured to (default), but at the cost of making the stats not particularly useful.
- It would probably also be useful for periodic reports to allow the logging system to keep track of when it last ran any reports, and to be able to automatically continue from there. If we used a database table for this we could add as many different analysers as we like over time, and each could collect their own report dates from this one data source.

</html>